FVPP: Hoareova logika i Dafny

Automatsko dokazivanje ispravnosti programa

Pripremio: izv. prof. dr. sc. Alan Jović

Ak. god. 2022./2023.

Sadržaj

- Dokazivanje ispravnosti programa
- Hoareova logika
  - Semantika programa
  - Jezik IMP
  - Jezik tvrdnji za Hoareovu logiku
  - Pravila prirodnog zaključivanja
  - Primjeri
- Programski alat Dafny

DOKAZIVANJE ISPRAVNOSTI PROGRAMA

Dokazivanje ispravnosti vs. provjera modela vs. ispitivanje

- **Provjera modela je jednostavnija od dokazivanja ispravnosti**
  - Provjeravaju se pojedinačna jednostavna svojstva programa
  - Ne iskazuju se potpuni zahtjevi (ugovori) o programu
  - Vrlo malo sustava se može dokazati direktno – zahtijeva se izgradnja modela
  - Dobro radi za kritični softver i hardver, slabije radi na podatkovno-intenzivnom i korisničkom softveru
- **Provjeru modela se lakše automatizira i sličnija je ispitivanju od dokazivanja ispravnosti**
  - Dokazivanje ispravnosti je mnogo temeljitije, teško se automatizira
- **Provjera modela pronalazi više nedostataka od ispitivanja**
  - Provjera modela prolazi **svim putovima** kroz **apstrahirani program**
  - Ispitivanje prolazi **nekim putovima** kroz **cjelokupni program**

Dokazivanje ispravnosti programa

- Za neki računalni program želimo **dokazati** da ispravno radi
  - Matematički dokaz na temelju poznavanja **koda programa** i postavljanja **uvjeta** na njegovo izvođenje u obliku **anotacija**
  - Primjer koda i uvjeta:

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = x - 2
  return y
}
```

Preduvjet

Uvjet po završetku programa (postuvjet)

Program

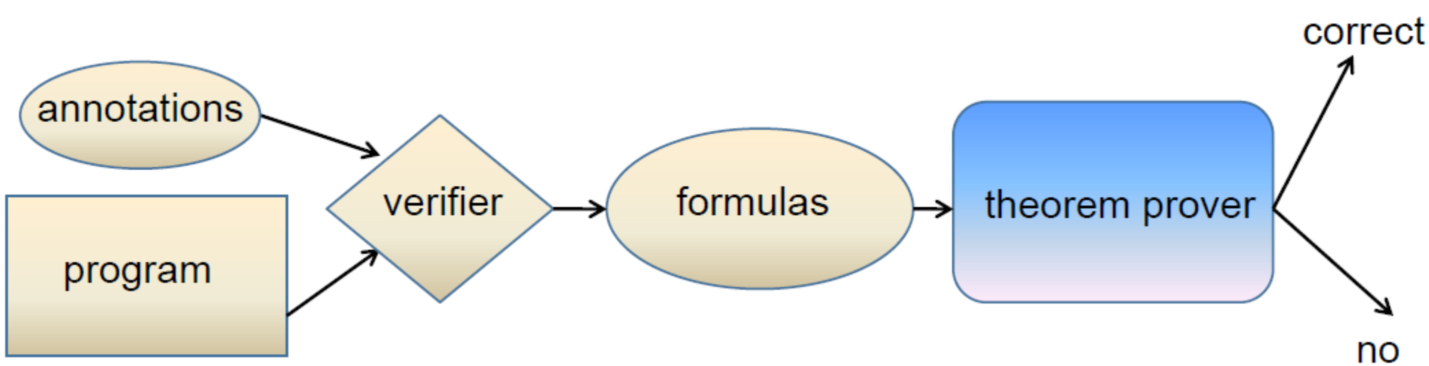
$$\forall x. \forall y. x > 0 \wedge y = x - 2 \rightarrow y > 0$$

Dokazivanje ispravnosti programa

- Anotacije piše programer ili analitičar programske potpore (engl. *software analyst*)
  - **Dodaju se izvornom kodu da bi izrazili svojstva koja se trebaju rasuditi o programu**
  - Tipični primjeri anotacija su one vezane uz **ugovor**:
    - **Preduvjeti** (engl. *precondition*) – opisuju nužna svojstva ulaznih vrijednosti u neki program
    - **Postuvjeti** (engl. *postcondition*) – opisuju što program treba napraviti (do čega treba dovesti)
    - **Invarijante** (engl. *invariants*) – opisuju svojstva koja trebaju vrijediti u svakom trenutku izvođenja programa
  - Ako anotacijsko svojstvo ne vrijedi, trebali bismo moći otkriti pogrešku u kodu programa

Dokazivanje ispravnosti programa

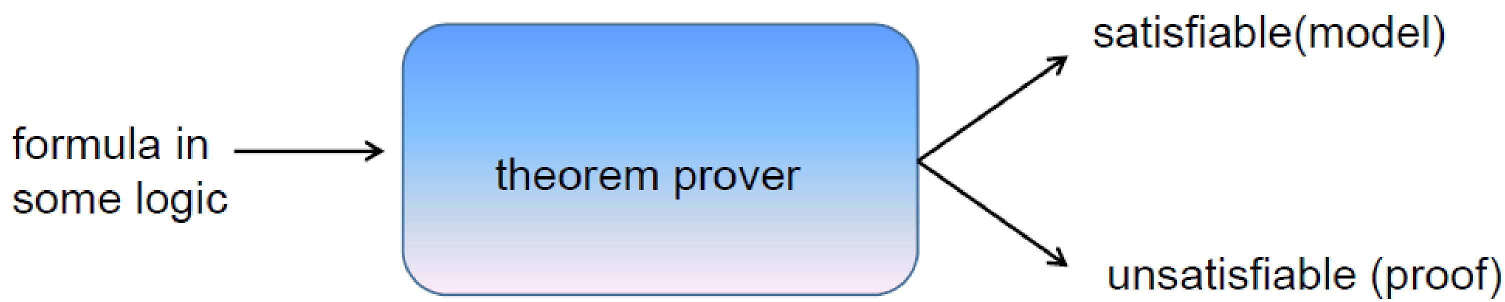
- Za automatsko dokazivanje ispravnosti programa razvijeni su programi za automatsko dokazivanje teorema (engl. *theorem prover*), ili jednostavnije „dokazivači" (engl. *prover*)



- Razlika u odnosu na dokazivanje formula u vremenskoj propozicijskoj logici: **stvarni programi imaju bitno složeniju matematičku teoriju** (logiku)

Dokazivanje ispravnosti programa

- **Procedura odlučivanja** (engl. *decision procedure*)
  - Algoritam koji odgovara na pitanje je li ulazna formula zadana u nekoj logici zadovoljiva ili ne
    - Npr. formula $x \leq y$ je zadovoljiva za $x = 0,\ y = 1$
    - Npr. formula $x \leq y \text{ i } x + 1 > y + 1$ nije zadovoljiva



  - U praksi, dokazivač nad stvarnim programima je vrlo složen algoritam koji se danas zasniva na **SAT-rješavačima** i **SMT-rješavačima** (tema ranijeg predavanja)

Dokazivanje ispravnosti programa

```
//: assume (x > 5)
def simple (Int x)
//: ensures y > 7
{
  val y = x + 2
  return y
}
```

- Što nam sve treba da dokažemo ispravnost?
  - Programski jezik
  - Izvođenje programa (i način za prikazati „kad god je program u nekom stanju...")
  - Jezik za anotacije
  - Način da to izkombiniramo

- Želimo izraziti i dokazati: „uvijek kada program primi kao ulaznu vrijednost *x*, takvu da je *x* > 5 i pokrenemo program, rezultat *y* će biti takav da vrijedi *y* > 7"
- Želimo striktnu formalnu definiciju kada je program ispravan. Zašto?
  - Zato da možemo izgraditi **automatski alat** koji će znati kada je taj cilj postignut
- Problem: **kako formalno specificirati programski jezik?**

Formalna semantika jezika

Vrste formalne semantike programskih jezika:

- Denotacijska semantika (engl. *Denotational Semantics*)
  - Značenje programa je definirano konstrukcijom matematičkog objekta (tzv. denotacija) koji opisuju značenje onoga što program izražava (npr. djelomična funkcija – funkcija nije definirana za sve ulazne parametre, kao što je dijeljenje cijelih brojeva)
  - Primjer: **Apstraktna interpretacija** (engl. *Abstract Interpretation*)
- Aksiomatska semantika (engl. *Axiomatic Semantics*)
  - Značenje programa je definirano kao učinak koji program ima na istinitost logičkih tvrdnji
  - Primjer: **Hoareova logika** (engl. *Hoare Logic*)
- Operacijska semantika (engl. *Operational Semantics*)
  - Značenje programa je definirano formalizacijom pojedinačnih koraka izračunavanja u programu
  - Primjer: **Označeni tranzicijski sustavi** (engl. *Labeled Transition Systems*)

Dokazivanje ispravnosti programa

- Primjer programskog jezika: **Java**
- **Java Language Specification** (JLS) za Javu 8 (LINK) opisuje semantiku Java programa
  - Dokument ima 780 stranica (malo bolje za Python 3 – „samo" 135 stranica)
  - 148 stranica samo za definiciju semantike izraza
  - 48 stranica za definiciju semantike pozivanja metoda
- Semantika Jave (i drugih jezika) je definirana samo prozno (za razliku od sintakse)
- Kako možemo napraviti tu semantiku formalnom?
- Trebamo matematički **model** računanja u programima – netrivijalno za postići!

HOAREOVA LOGIKA

Hoareova logika

- Sir Charles Antony Richard (Tony) Hoare, r. 1934.
- **Hoareova logika** (1969.+) je formalni sustav koji koristi skup deduktivnih pravila zaključivanja za rasuđivanje nad računalnim programima
- Zasniva se na **Hoareovoj trojci** (tripletu):

  $\{P\}\ C\ \{Q\}$

  gdje su P i Q tvrdnje a C je naredba programa
- P se naziva **preduvjet** (engl. *precondition*), a Q **postuvjet** (engl. *postcondition*)
- Semantika Hoareove trojke:
  - Ako je **ispunjen preduvjet** pokreće se **naredba** koja, ako završi, dovodi do **ispunjavanja postuvjeta**

Značaj Hoareove logike

- Razvija se zadnjih 50 godina s težnjom automatske provjere ispravnosti računalnih programa
- Na nju se nadograđuju brojna proširenja za provjeru ispravnosti, npr.
  - Separacijska logika (engl. *separation logic*)
  - Združena logika (engl. *bunched logic*)
  - Hoareova logika za vremenska ograničenja (engl. *Hoare logic for time constraints*)
  - Relacijska Hoareova logika (engl. *relational Hoare logic*)
- Hoareova logika je korištena za dokazivanje ispravnosti većeg broja korisnih algoritama i programa, npr.
  - *Quicksort*
  - *Timsort*
  - Linearni programi (npr. za linearne diferencijalne jednadžbe i sl.)
  - Kvantni programi

Potpuna i djelomična ispravnost

- Hoareova logika ima precizniju **aksiomatsku semantiku** koja **zahtijeva djelomičnu ispravnost programa**
- **Djelomična ispravnost**
  - Ako program završi izvođenje počevši od ispunjenih određenih preduvjeta tada on ispunjava željene postuvjete. Nema garancije da se program treba završiti
- **Potpuna ispravnost** (nije zahtijevana Hoareovom logikom)
  - Program koji počinje od ispunjenih preduvjeta uvijek završava i ispunjava željene postuvjete
- Kako bismo formalizirali provjeru ispravnosti, trebamo:
  1. Objasniti formalnu semantiku,
  2. Dati primjer jezika (sintakse i semantike) na kojem možemo provjeriti ispravnost
  3. Navesti pravila zaključivanja u tom jeziku u kontekstu Hoareove logike

Aksiomatska semantika

- Aksiomatska semantika sastoji se od:
  - Jezika za navođenje **tvrdnji** (u praksi: anotacija) (engl. *assertions*) o programima
  - Pravila za ustanovljavanje istinitosti o tvrdnjama
- Neke tipične tvrdnje:
  - „Ovaj program završava s izvođenjem"
  - „Ako ovaj program završava s izvođenjem, varijable *x* i *y* imaju istu vrijednost tijekom izvršavanja programa"
  - „Pristup elementima polja je uvijek unutar ograničenja veličine polja"
- Tipični jezici za iskaz tvrdnji:
  - **Logika predikata prvoga reda**
  - Druge vrste logike (npr. vremenska), FOPL s proširenjima (teorijama)
  - Drugi specijalni specifikacijski jezici (npr. Z, JML)

IMP – jednostavni imperativni jezik

- Uvedimo primjer jezika na kojem možemo ustanoviti ispravnost
  - bitno pojednostavljeni imperativni programski jezik naziva „IMP”
- Sintaksa jezika IMP:
  - $n \in \mathbb{Z}$ – cijeli brojevi
  - *true*, *false* $\in$ **B** – Boolean
  - $x, y \in Vars$ – varijable programa
  - $e \in \boldsymbol{Aexp}$ – aritmetički izrazi
  - $b \in \boldsymbol{Bexp}$ – Booleovi izrazi
  - $c \in \boldsymbol{Com}$ – naredbe

IMP – jednostavni imperativni jezik

- Sintaksa aritmetičkih izraza ***Aexp***
  - $e ::= n,\ n \in \mathbb{Z}$

    $\mid x,\ x \in Vars$

    $\mid e_1 + e_2$

    $\mid e_1 - e_2$

    $\mid e_1 * e_2$

- Varijable se ne deklariraju prije korištenja
- Sve varijable su cjelobrojnog tipa
- Izrazi nemaju nikakve nuspojave (ne utječu na ostale varijable i izraze)

IMP – jednostavni imperativni jezik

- Sintaksa Booleovih izraza ***Bexp***
  - $b ::= true$

    $| \; false$

    $| \; e_1 = e_2$, **za** $e_1, e_2 \in Aexp$

    $| \; e_1 \leq e_2$, **za** $e_1, e_2 \in Aexp$

    $| \; \neg b \in Bexp$

    $| \; b_1 \wedge b_2$, **za** $b_1, b_2 \in Bexp$

    $| \; b_1 \vee b_2$, **za** $b_1, b_2 \in Bexp$

IMP – jednostavni imperativni jezik

- Sintaksa naredbi ***Com***
  - $c ::= skip$

    $| \; x = e$

    $| \; c_1; c_2,$ **za** $c_1, \; c_2 \in Com$

    $| \; \text{if } b \text{ then } c_1 \text{ else } c_2$

    $| \; \text{while } b \text{ do } c$

- Napomena: jeziku nedostaju reference, funkcijski pozivi, druge vrste petlji, polja, složenije strukture podataka ...

IMP – jednostavni imperativni jezik

- Semantika jezika IMP
  - Značenje izraza e u IMP-u ovisi o vrijednostima varijabli, tj. o trenutačnom stanju
  - Stanje je u svakom trenutku predstavljeno funkcijom od Vars u $\mathbb{Z}$
  - Skup svih stanja je $Q = \text{Vars} \to \mathbb{Z}$
  - Koristimo malo q da izrazimo pojedinačna stanja iz Q

Semantika IMP-a: presude

- Pišemo <e, q> $\Downarrow$ n u značenju da se izraz e **evaluira** kao n u stanju q.
- Formula <e, q> $\Downarrow$ n naziva se **presuda**
  - Presuda je izjava o odnosu između e, q i n
- U ovom slučaju presuda se može razmatrati kao funkcija dvaju argumenata: e i q koja vraća n
- Ovakva formulacija naziva se **prirodna operacijska semantika** (engl. *natural operational semantics*)
  - Presuda dovodi u odnos izraz e i njegovo značenje n

Pravila zaključivanja

- Pravila evaluiranja (engl. *evaluation rules*) izraza izražavamo u obliku **pravila zaključivanja** (engl. *inference rules*) za naše presude
- Pravilo zaključivanja:

  $\dfrac{F_1 \ldots F_n}{G}$, gdje H

  definira relaciju između presuda $F_1 \ldots F_n$ i G

- Presude $F_1 \ldots F_n$ su premise pravila zaključivanja
- Presuda G je zaključak pravila
- Formula H je sporedni uvjet (engl. *side condition*) pravila
- Ako je n=0 onda je pravilo aksiom i crta se može ispustiti

Pravila zaključivanja za IMP *Aexp*

- Uglavnom po jedno pravilo zaključivanja za svaki konstrukt jezika:

$<n, q> \Downarrow n$ - aksiom

$<x, q> \Downarrow q(x)$ - aksiom

$$\frac{<e_1,q>\Downarrow n_1 \quad <e_2,q>\Downarrow n_2}{<e_1+e_2,q>\Downarrow (n_1+n_2)}$$ - adicijsko pravilo

$$\frac{<e_1,q>\Downarrow n_1 \quad <e_2,q>\Downarrow n_2}{<e_1-e_2,q>\Downarrow (n_1-n_2)}$$ - suptrakcijsko pravilo

$$\frac{<e_1,q>\Downarrow n_1 \quad <e_2,q>\Downarrow n_2}{<e_1*e_2,q>\Downarrow (n_1*n_2)}$$ - multiplikacijsko pravilo

- Pravila su definirana na osnovi strukture izraza – **strukturalna operacijska semantika**

Pravila zaključivanja za IMP *Bexp*

$<true, q> \Downarrow true$ - aksiom

$<false, q> \Downarrow false$ - aksiom

$$\frac{<e_1,q>\Downarrow n_1 \quad <e_2,q>\Downarrow n_2}{<e_1=e_2,q>\Downarrow(n_1=n_2)}$$

$$\frac{<e_1,q>\Downarrow n_1 \quad <e_2,q>\Downarrow n_2}{<e_1\leq e_2,q>\Downarrow(n_1\leq n_2)}$$

$$\frac{<b_1,q>\Downarrow t_1 \quad <b_2,q>\Downarrow t_2}{<b_1\, i\, b_2,q>\Downarrow(t_1\, i\, t_2)}, i = \wedge$$

Kako čitati pravila zaključivanja

- **Unaprijed**
  - Ako znamo da presude u premisama vrijede onda možemo zaključiti i da presuda u zaključku vrijedi, npr.

$$\frac{<2,q>\Downarrow 2 \qquad <3,q>\Downarrow 3}{<2*3,q>\Downarrow 6}$$

- **Unazad**, **rasuđivanjem inverzijom**
  - Npr. ako želimo evaluirati $e_1 + e_2$ uočavamo da zaključak $e_1 + e_2 \Downarrow n$ mora biti **adicijsko** pravilo (samo je jedno takvo)
  - Rekurzivno idemo unatrag po pravilima, u svakom koraku primijenimo jedno pravilo dok ne dođemo do najosnovnije evaluacije (u ovom slučaju određivanje n_1 i n_2)

Kako čitati pravila zaključivanja

- 1. korak:

$$\frac{<\mathrm{x}, \{x \mapsto 3, y \mapsto 2\}> \Downarrow 3 \qquad <2 * \mathrm{y}, \{x \mapsto 3, y \mapsto 2\}> \Downarrow 4}{<\mathrm{x} + (2 * \mathrm{y}), \{x \mapsto 3, y \mapsto 2\}> \Downarrow 7}$$

- 2. korak:

$$\frac{<\mathrm{x}, \{x \mapsto 3, y \mapsto 2\}> \Downarrow 3 \qquad \dfrac{\begin{array}{c} <y, \{x \mapsto 3, y \mapsto 2\}> \Downarrow 2 \\ <2, \{x \mapsto 3, y \mapsto 2\}> \Downarrow 2 \end{array}}{<2 * \mathrm{y}, \{x \mapsto 3, y \mapsto 2\}> \Downarrow 4}}{<\mathrm{x} + (2 * \mathrm{y}), \{x \mapsto 3, y \mapsto 2\}> \Downarrow 7}$$

- Tu stajemo, jer smo došli do aksioma: $<2, q> \Downarrow 2$ i $<y, q> \Downarrow 2$

Semantika IMP-a: naredbe

- Evaluacija naredbe (*Com*) ima nuspojave, ali nijedan izravni rezultat
- Rezultat naredbe *c* pokrenute u predstanju *q* je **prijelaz** iz *q* u poststanje *q*' :

$$q \overset{c}{\to} q'$$

- Formalizacija evaluacije naredbi može se napraviti pomoću **označenih tranzicijskih sustava** (engl. *labeled transition systems*)

Označeni tranzicijski sustav

- Označeni tranzicijski sustav (kraće: LTS) je struktura

  LTS = (*Q*, *Act*, →) gdje je:

  Q – skup **stanja** sustava

  Act – skup **akcija**

  $\rightarrow \subseteq Q \times Act \times Q$ – relacija prijelaza (engl. *transition relation*)

- Pišemo $q \xrightarrow{a} q'$ za $(q, a, q') \in \rightarrow$

- Potrebno je definirati semantiku za sve naredbe (akcije) *a* LTS-a za jezik IMP, pri čemu razmatramo promjene stanja *q* kao rezultat djelovanja naredbi

Semantika IMP-a: naredbe putem LTS

- $q \xrightarrow{skip} q$ (za skip) $\dfrac{q \xrightarrow{c_1} q' \quad q' \xrightarrow{c_2} q''}{q \xrightarrow{c_1\ ;\ c_2} q''}$ (za c_1 ; c_2)
- $\dfrac{<e,q>\Downarrow n}{q \xrightarrow{x:=e} q \mathbin{++} \{x \mapsto n\}}$ (za x = e)
- $\dfrac{<b,q>\Downarrow true \quad q \xrightarrow{c_1} q'}{q \xrightarrow{\text{if } b \text{ then } c_1 \text{ else } c_2} q'}$ $\dfrac{<b,q>\Downarrow false \quad q \xrightarrow{c_2} q'}{q \xrightarrow{\text{if } b \text{ then } c_1 \text{ else } c_2} q'}$ (za if b then c_1 else c_2)
- $\dfrac{<b,q>\Downarrow false}{q \xrightarrow{while\ b\ do\ c} q}$ $\dfrac{<b,q>\Downarrow true \quad q \xrightarrow{c} q' \quad q' \xrightarrow{while\ b\ do\ c} q''}{q \xrightarrow{while\ b\ do\ c} q''}$ (za while b do c)

Semantika IMP-a: prijelazi

- Tvrdnje za jezik IMP mogu se iskazati u obliku
- {A} *c* {B} sa značenjem da:
- ako A vrijedi u stanju q i $q \xrightarrow{c} q'$ onda B vrijedi u q'
- A je preduvjet, a B je postuvjet
- Primjer tvrdnje:

$$\{ \mathrm{y} \leq \mathrm{x} \}\ \mathrm{z} := \mathrm{x};\ \mathrm{z} := \mathrm{z} + 1\ \{ \mathrm{y} < \mathrm{z} \}$$

- Ova tvrdnja je validna (vrijedi)

- Tvrdnje ovog tipa za jezik IMP su Hoareove trojke (tripleti) koje su djelomično ispravne

Jezik tvrdnji za Hoareovu logiku

- Za iskaz tvrdnje u Hoareovoj logici koristi se **predikatna logika prvoga reda** zajedno s izrazima u **IMP-u**:
- A :: = *true* | *false* | $e_1 = e_2$ | $e_1 \geq e_2$ | $A_1 \wedge A_2$ | $A_1 \vee A_2$ | $A_1 \Rightarrow A_2$ | $\forall x$.A | $\exists x$.A
- Implicitno, sve IMP varijable imaju raspon cijelih brojeva
- Svi IMP-ovi Booleovi izrazi su isto tako tvrdnje A.

- Značenje aksiomatskog jezika:
  - Oznaka $q \vDash$ A znači da tvrdnja A vrijedi u danom stanju q.
  - Presuda $\vDash$ je definirana induktivno na strukturi tvrdnji
  - Oznaka $\vDash$ A znači da tvrdnja A vrijedi u bilo kojem stanju, uvijek je istinita

Semantika tvrdnji za Hoareovu logiku

- $q \vDash true$ Stanje q uvijek vrijedi
- $q \vDash e_1 = e_2$ ako i samo ako $<e_1, q>\Downarrow \, = \, <e_2, q>\Downarrow$
- $q \vDash e_1 \geq e_2$ ako i samo ako $<e_1, q>\Downarrow \, \geq \, <e_2, q>\Downarrow$
- $q \vDash A_1 \wedge A_2$ ako i samo ako $q \vDash A_1$ i $q \vDash A_2$
- $q \vDash A_1 \vee A_2$ ako i samo ako $q \vDash A_1$ ili $q \vDash A_2$
- $q \vDash A_1 \Rightarrow A_2$ ako i samo ako $q \vDash A_1$ implicira $q \vDash A_2$
- $q \vDash \forall x.\mathrm{A}$ ako i samo ako $\forall n \in \mathbb{Z}.\ q[x{:=}n] \vDash \mathrm{A}$
- $q \vDash \exists x.\mathrm{A}$ ako i samo ako $\exists n \in \mathbb{Z}.\ q[x{:=}n] \vDash \mathrm{A}$

Formalna definicija djelomične i potpune ispravnosti

- Djelomična ispravnost:
- $\vDash \{\mathrm{A}\}\ c\ \{\mathrm{B}\}$ ako i samo ako $\forall q \in Q.\ \forall q' \in Q.\ q \vDash \mathrm{A} \wedge q \xrightarrow{c} q' \Rightarrow q' \vDash \mathrm{B}$

- Potpuna ispravnost:
- $\vDash [\mathrm{A}]\ c\ [\mathrm{B}]$ ako i samo ako $\forall q \in Q.\ q \vDash \mathrm{A} \Rightarrow \exists q' \in Q.\ q \xrightarrow{c} q' \wedge q' \vDash \mathrm{B}$

- *q* je stanje, definira vrijednosti varijabli
- {A} *c* {B} – Hoareova trojka
- $q \vDash \mathrm{F}$ – u stanju q vrijedi formula F

- **Važan rezultat:** sada se može formalno dokazati da je program ispravan u smislu operacijske semantike
- **Problem:** program se mora pokrenuti da bi se dokazala njegova ispravnost – treba nam automatsko dokazivanje teorema bez pokretanja programa

Pravila prirodnog zaključivanja za propozicijsku logiku

- U propozicijskoj logici vrijede pravila prirodnog zaključivanja koja smo uveli na 2. predavanju
- Ona se mogu pisati s oznakom $\vdash A$ kada je A moguće zaključiti iz inicijalnih aksioma (A je dedukcija)
- Neka od pravila su (vidjeti 2. predavanje, zadnji slajd, za njih sve):

$$\frac{\vdash A \qquad \vdash B}{\vdash A \text{ i } B}$$ (uvođenje konjunkcije)

$$\frac{\vdash A}{\vdash A \text{ ili } B}$$ *ili* = $\vee$ (uvođenje disjunkcije)

$$\frac{\vdash A \Rightarrow \mathrm{B} \qquad \vdash A}{\vdash B}$$ (modus ponens)

...

Pravila prirodnog zaključivanja za predikatnu logiku

- U predikatnoj logici prvog reda vrijede i dodatna pravila zaključivanja:

$$\frac{\vdash A[a/x]}{\vdash \forall x.\, A}$$ uvođenje univerzalnog kvantifikatora

$$\frac{\vdash \forall x.\, A}{\vdash A[e/x]}$$ eliminacija univerzalnog kvantifikatora

$$\frac{\vdash A[e/x]}{\vdash \exists x.\, A}$$ uvođenje egzistencijskog kvantifikatora

$$\frac{\vdash \exists x.\, A \qquad \begin{array}{c}\vdash A[a/x] \\ \vdots \\ \vdash B\end{array}}{\vdash B}$$ eliminacija egzistencijskog kvantifikatora

- pri čemu $A[a/x]$ označava operaciju supstitucije (unifikacije) – zamjenu svih slobodnih pojava varijable *x* u A s vrijednosti *a*, uz dodatni uvjet da je ***a* svježa vrijednost** (ne pojavljuje se nigdje drugdje)
- $A[e/x]$ označava zamjenu svih slobodnih pojava varijable *x* u A s **izrazom** e

Pravila prirodnog zaključivanja za Hoareovu logiku

- Po jedno pravilo za svaki sintaksni konstrukt jezika IMP (uzimajući u obzir predikatnu logiku prvoga reda:

- $\vdash \{A\}\ \mathrm{skip}\ \{A\}$ – **aksiom skip** – prazna naredba ne mijenja stanje programa
- $\vdash \{A[e/x]\}\ x{:=}e\ \{A\}$ – **aksiom pridruživanja:** svaka slobodna pojava varijable *x* zamijenjena je izrazom e
- $\dfrac{\vdash \{A\}\, c_1\, \{B\} \qquad \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1\,;\, c_2\, \{C\}}$ – **pravilo kompozicije**
- $\dfrac{\vdash \{A \land b\}\ c_1\ \{B\} \qquad \vdash \{A \land \neg b\}\ c_2\ \{B\}}{\vdash \{A\}\ \mathrm{if}\ b\ \mathrm{then}\ c_1\ \mathrm{else}\ c_2\ \{B\}}$ – **pravilo uvjeta**
- $\dfrac{\vdash \{I \land b\}\ c\ \{I\}}{\vdash \{I\}\ \mathrm{while}\ b\ \mathrm{do}\ c\ \{I \land \neg b\}}$ – **pravilo petlje while**

- I dodatno **pravilo posljedice** (engl. *rule of consequence*):
- $\dfrac{\vdash A' \Rightarrow A \quad \vdash \{A\}\, c\, \{B\} \quad \vdash B \Rightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$

Invarijanta petlje

- *I* – invarijanta petlje je formula koja:
  - vrijedi **prije ulaza u petlju prvi put** i u **svim stanjima koja zadovoljavaju preduvjet**
  - njezina vrijednost se **održava (vrijedi) nakon svake iteracije petlje i na kraju izvođenja petlje**

Primjer: jednostavna petlja

$$\frac{\vdash \{\mathtt{I} \wedge b\}\ \mathtt{c}\ \{\mathtt{I}\}}{\vdash \{\mathtt{I}\}\ \mathtt{while}\ b\ \mathtt{do}\ c\ \{\mathtt{I} \wedge \neg b\}}$$

- Želimo zaključiti da vrijedi:

  $\vdash \{x \leq 0\}$ while $x \leq 5$ do $x := x + 1$ $\{x = 6\}$

- Trebamo iskoristiti pravilo za while s invarijantom petlje $I \equiv x \leq 6$

$$\frac{\vdash \{x \leq 6 \wedge x \leq 5\}\ x := x + 1\ \{x \leq 6\}}{\vdash \{\mathrm{x} \leq 6\}\ \mathtt{while}\ \ \mathrm{x} \leq 5\ \mathtt{do}\ \ \mathrm{x} := \mathrm{x} + 1\ \{\mathrm{x} \leq 6 \wedge \mathrm{x} > 5\}}$$

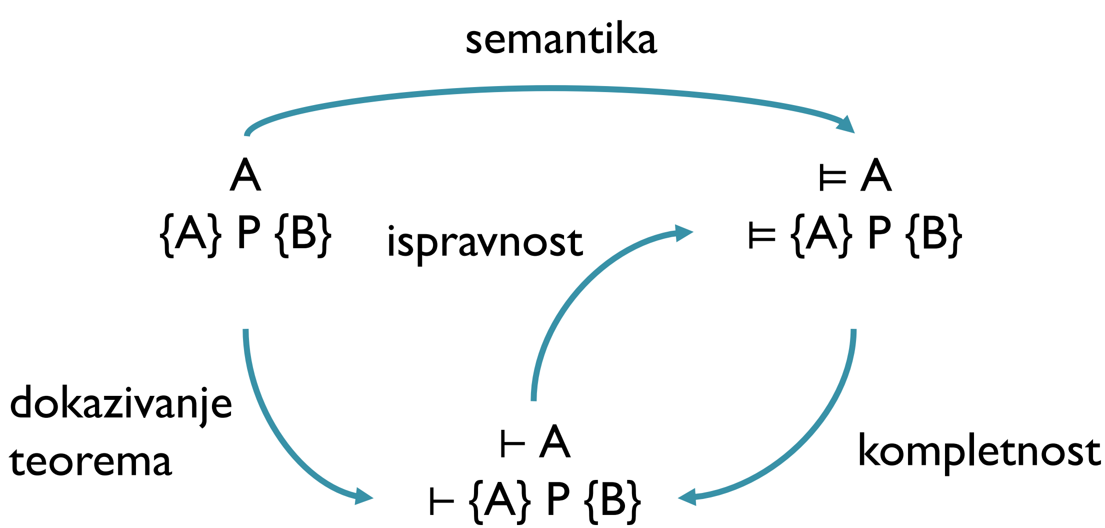
$$\frac{\vdash A' \Rightarrow A \quad \vdash \{A\}\ c\ \{B\} \quad \vdash B \Rightarrow B'}{\vdash \{A'\}\ c\ \{B'\}}$$

- Potom treba primijeniti pravilo posljedice da bismo dokazali tvrdnju:

$$\frac{\vdash x \leq 0 \Rightarrow x \leq 6 \qquad \vdash x \leq 6 \wedge x > 5 \Rightarrow x = 6 \qquad \vdash \{x \leq 6\}\ \mathtt{while}\ \ \ldots\ \{x \leq 6 \wedge x > 5\}}{\vdash \{\mathrm{x} \leq 0\}\ \mathtt{while}\ldots\ \{\mathrm{x} = 6\}}$$

Zaključno o Hoareovoj logici

- Imamo jezik za izražavanje tvrdnji o programima
- Znamo kada je takva tvrdnja istinita
- Imamo i simboličku metodu (pravila prirodnog zaključivanja) kako bismo izveli tvrdnje



Zaključno o Hoareovoj logici

- Hoareova logika omogućuje rasuđivati o programima
- Primjena pravila prirodnog zaključivanja u Hoareovoj logici nije jednostavna, ali se može automatizirati; pritom su problemi (ipak riješivi):
  - Kada primijeniti pravilo posljedice?
  - Koje invarijante koristiti za while petlju?
  - Kako dokazati implikacije uključene u pravilo posljedice?
- Ako je IMP program donekle složeniji dokaz može biti **dug**
- Što ako imamo potrebe za rasuđivanje o još složenijim programima nego što nam dozvoljava sintaksa i semantika IMP-a?
  - Trebaju nam teorije za zaključivanje sa složenijim strukturama podataka (polja, skupovi, liste)
  - Trebaju nam teorije za zaključivanje o zauzimanju memorije
- Ovo izlazi izvan okvira klasične Hoareove logike

PROGRAMSKI ALAT DAFNY

Programski alat Dafny

- Programski alat Dafny (2013.+) sastoji se od:
  - **imperativnog objektnog jezika za anotirane programe** i
  - **verifikatora koji prevodi anotirane programe u međujezik za platformu .NET**
- https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/
- https://dafny.org/
- Prevoditelj **statički** (statički verifikator) provjerava:
  - odsustvo pogrešaka pri izvođenju (engl. *runtime errors*)
  - mogućnost završetka petlji i poziva metoda
  - ispravnost korisnički definiranih ugovora

- Ref.: Rustan Leino, Michal Moskal, Co-Induction Simply: Automatic Co-Inductive Proofs in a Program Verifier, MSR-TR-2013-49 | July 2013

Programski alat Dafny

- Kao objektni jezik, Dafny podržava:
  - generičke razrede
  - reference na objekte, dinamičku alokaciju memorije
- Jezik podržava specifikaciju i verifikaciju:
  - **ugovore metoda** – preduvjete, postuvjete, uvjete okvira (engl. *frame conditions*) – klauzule koje izražavaju one objekte koji se smiju mijenjati
  - **invarijante petlji** i *inline* tvrdnje (unutar koda)
  - nepromijenjive tipove podataka (engl. *immutable types*): skupove, sekvence, algebarske tipove podataka
  - i dr.

Alat Boogie

- Alat **Boogie** (2008.+, Microsoft) je verifikator međujezika koji generira Dafny
- Boogie je potpuno automatizirani deduktivni verifikacijski sustav koji generira uvjete za verifikaciju koji se šalju **SMT-rješavaču** na provjeru (defaultni je **Z3**)
- SMT-rješavači znaju rasuđivati o složenijim teorijama nego što omogućuje jezik IMP i Hoareova logika
- Boogie koristi **semantiku najslabijeg preduvjeta**

Algoritam za verifikaciju (generiranje uvjeta verifikacije)

- Ideja: propagirati postuvjet **unazad** kroz program kako bismo došli do preduvjeta koji treba biti ispunjen – generiranje uvjeta za uspješnu verifikaciju
- Od {*A*} P {*B*}:

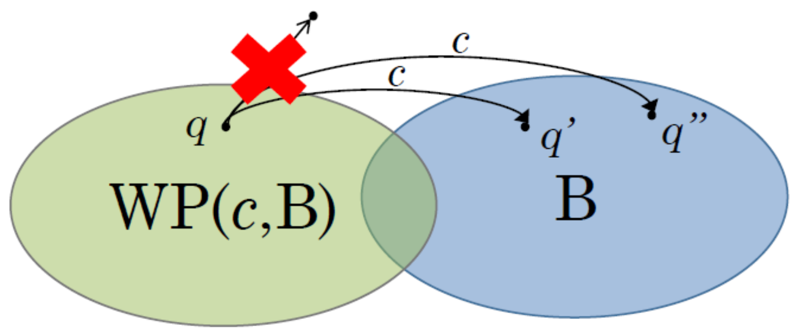
  generira se formula $A \Rightarrow F(P, B)$, gdje je $F(P, B)$ formula koja opisuje

  početna stanja za program koji će završiti u *B*
- Generiranje uvjeta za verifikaciju (engl. *Verification Condition Generation*, **VCG**)

- Propagacija unazad $F(P, B)$ može se formalizirati putem **najslabijih preduvjeta** (engl. *weakest precondition*)

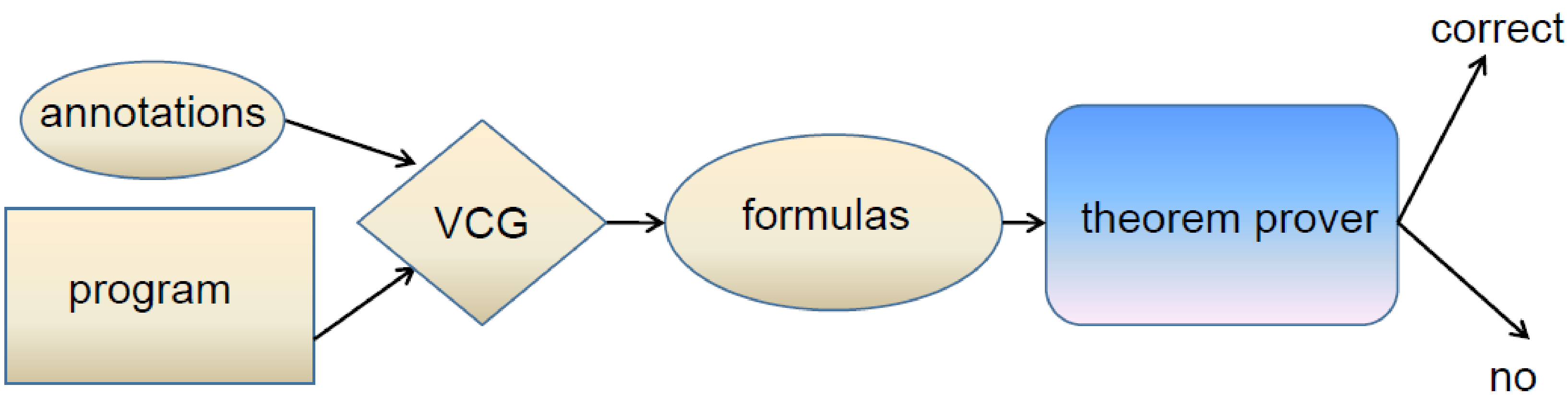
Semantika najslabijeg preduvjeta

- Dijkstra je predložio pristup implementaciji verifikacije programa koji se zove semantika najslabijeg preduvjeta
- **Najslabiji preduvjet WP(c, B) vrijedi za svako stanje *q* čija stanja c-sljedbenika sva zadovoljavaju B:**
- $q \vDash \mathrm{WP}(c,\mathrm{B})$ ako i samo ako $\forall q' \in Q.\ q \xrightarrow{c} q' \Rightarrow q' \vDash \mathrm{B}$



- Izračuna se WP(*P*,B) rekurzivno od kraja prema strukturi programa *P*.
- Ovaj pristup se razlikuje od Hoareovog izvornog pristupa budući da se inzistira na **potpunoj ispravnosti** umjesto na djelomičnoj ispravnosti uz **uvjet da je na početku samo zadan postuvjet**

Primjena SMT-rješavača



- VCG generira sveukupnu formulu ograničenja (koja se može razmatrati kao više pojedinačnih formula), a koja predstavlja naš program (neku konkretnu funkciju) za koji se treba utvrditi ispravnost
- Ovdje je *theorem prover* SMT-rješavač koji utvrđuje je li navedena formula ispravna

Dafny – deklaracija metode

- Primjer **metode**:

```
method Abs(x: int) returns (y: int)
{
  if (x < 0) { return-x; }
  else { y := x; }
}
```

- Nužno se eksplicitno navodi **povratna vrijednost (returns)** metode

Dafny – funkcije i predikati

- **Funkcije** su metode koje:
  - **ne smiju imati nuspojave** (sve što se događa, događa se samo unutar funkcije)
  - **uvijek moraju završiti** (potpuna ispravnost)
- Oba svojstva provjerava verifikator
- Mogu se koristiti unutar specifikacije
- **Predikati** su one funkcije koje vraćaju tip podatka bool

Dafny – preduvjeti i postuvjeti

- Preduvjeti i postuvjeti deklariraju se pomoću ključnih riječi requires i ensures
- Primjer:

```
method MultipleReturns (x: int, y: int)
returns(more: int, less: int)
  requires 0 < y
  ensures less < x < more
{
  more := x + y;
  less := x -y;
}
```

Dafny – rad s petljama

- Dafny ne zna za vrijeme prevođenja koliko puta će se petlja while izvesti
- Ipak, verifikator treba razmotriti **sve moguće putove** kroz program
- **Invarijante petlji služe da bi verifikator eliminirao sve petlje u programu koristeći indukciju**
- Invarijantu petlji nužno navodi programer / analitičar programa
- Podsjetimo: invarijanta petlji je **formula** koja:
  - **vrijedi prije ulaza u petlju prvi put**
  - **njezina vrijednost se održava (vrijedi) nakon svake iteracije petlje**

Dafny – rad s petljama – primjer

```
method computeFib(n: nat) returns (m: nat)
  ensures m == Fibonacci(n)
{
  var i:= 0;
  var k := 1;
  m := 0;
  while(i < n)
    invariant 0 <= i <= n
    invariant k == Fibonacci(i+1) && m == Fibonacci(i)
  {
    m, k := k, m + k;
    i := i + 1;
  }
}
```

```
function Fibonacci(n: nat): nat
  decreases n
{
  if n < 2 then n else Fibonacci(n-2) + Fibonacci(n-1)
}
```

Dafny – uvjeti okvira

- Funkcije i metode trebaju specificirati **memorijski otisak** – lokacije kojima mogu pristupiti i/ili mijenjati
- Skup memorijskih lokacija naziva se **okvir** (engl. *frame*)
- Uvjeti okvira:
  - $\mathbf{reads}\ \mathrm{S}$ – specificira da **funkcija** čita samo lokacije u okviru S
  - $\mathbf{modifies}\ \mathrm{S}$ – specificira da **metoda** mijenja samo lokacije u okviru S
- Funkcije smiju čitati samo one lokacije koje su specificirane s njihovim reads klauzulama (anotacijama)
- Metode smiju pristupiti bilo kojoj lokaciji ali smiju mijenjati samo one lokacije koje specificiraju njihove $\mathrm{modifies}$ klauzule

Dafny – uvjeti okvira – primjer

```
predicate sorted(a: array<int>)
  requires a != null
  reads a
{
  forall j, k :: 0 <= j < k < a.Length==> a[j] <= a[k]
}

method BinarySearch(a: array<int>, key: int)
returns(index: int)
  requires a != null && sorted(a)
  ensures...
{
  ...
}
```

Zaključak

- Prikazane su osnove dokazivanja ispravnosti programa korištenjem pravila zaključivanja u Hoareovoj logici
- Programski alat Dafny implementira zaključivanje koje se jednim dijelom temelji na Hoareovoj logici
  - značajna nadogradnja Hoareove logike i korištenje SMT-rješavača za automatsko zaključivanje
- Preporuke za daljnje učenje:
  - proučiti detaljnije kako je implementiran **Dafny**
  - proučiti **separacijsku logiku** i njezinu primjenu za verifikaciju programa sa zahtjevnijim memorijskim strukturama
  - proučiti alternativne notacije za specifikacije, npr. **JML**